Napredna grafika - kul forice

Vladimir Milenković

Matematička gimnazija

NEDELJA INFORMATIKE v5.0

21. decembar 2018.

Zašto sve ovo? [DEMO]

- Zato što se u 150 linija može proizvesti nešto ovako (i to nije u Scratchu!): Kul zec
- A u malo više linija, nešto ovako: Kul roboti
- I ono što je interesantno je to da za ovako kratko vreme jednog predavanja možemo shvatiti suštinu koda koji radi ove stvari.

Čime ćemo se baviti danas?

- Ray casting
- Ray tracing
- Ray marching (Spoiler alert: Ovo je baš kul)

Zraci u stvarnosti

- U prirodu, izvor svetlosti emituje zrake koji putuju do neke površine koja ih zaustavi.
- Možemo reći da je zrak snop fotona koji idu po istoj putanji.
- Bilo koja kombinacija sledeće 4 stvari može da se desi svakom zraku:
  - Apsorpscija
  - Refleksija
  - Refrakcija (prelamanje)
  - Fluoroscencija

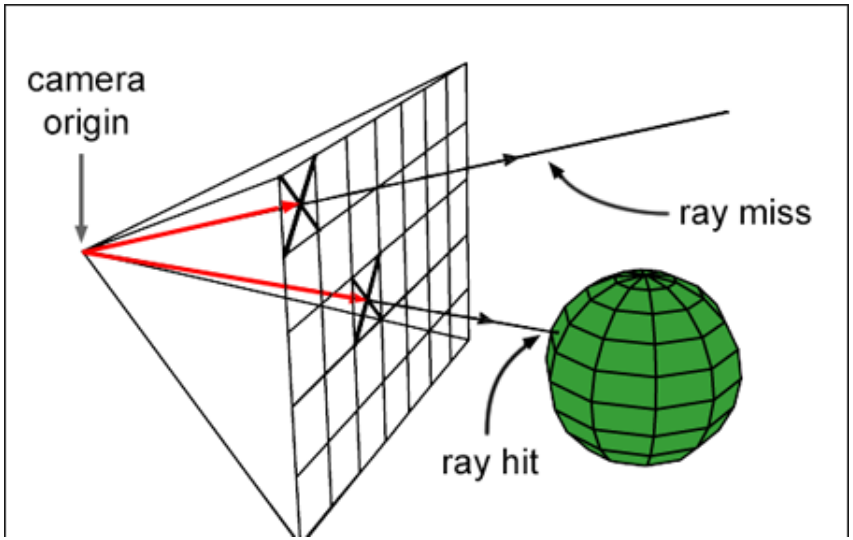
Kompjuterska grafika

- Poenta svih ovih algoritama je da pokušaju da, što je tačnije moguće, proizvedu isti efekat kao da mi to gledamo svojim očima.
- To znači da smemo malo da lažiramo fiziku ako se to ne vidi, što će biti vrlo korisno.
- Zlatno pravilo grafike - tačno je ako ne vidimo da nije :)

Ideja ray casting-a

- Prvi algoritam nalik na ray casting - 1968.
- Kroz svaki piksel ekrana pustimo zrak, i gledamo gde će taj zrak završiti.
- Tražimo preseke sa svim objektima na sceni, i gledamo najbliži presek - to je boja koju dodeljujemo našem zraku.
- Implementacija ovog algoritma je veoma laka
- Sledi kratka demonstracija

Ray casting - algoritam



Ray casting - pseudokod

```
select an eye point and a screen plane

FOR every pixel in the screen plane
    determine the ray from the eye through the pixel's centre
    FOR each object in the scene
        IF the object is intersected by the ray
            IF the intersection is the closest (so far) to the eye
                record intersection point and object
            END IF ;
        END IF ;
    END FOR ;
    set pixel's colour to that of the object at the closest intersection point
END FOR ;
```

Wolfenstein 3D

Pros and Cons

- Koja je složenost ovog algoritma?
- Kada je ova složenost dovoljno dobra, a kada nije?
- Koje stvari postoje u realnom svetu, a ovim algoritmom ih ne prikazujemo?
- Paralelizacija?

Šta je Ray tracing?

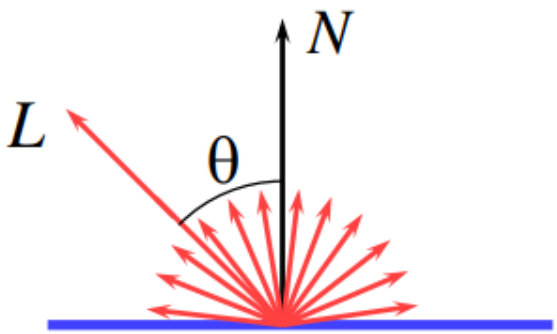
- Unapređeni Ray casting
- Pokušavamo da se pobrinemo oko stvari koje nisu već obuhvaćene, a doprineće realnom izgledu slike

Šta je Ray tracing?

- Unapređeni Ray casting
- Pokušavamo da se pobrinemo oko stvari koje nisu već obuhvaćene, a doprineće realnom izgledu slike
- Neke od tih stvari su:
  - Izvor svetlosti nezanemarljivih dimenzija
  - Senčenje
  - Osvetljenje
  - Različite površine objekata

Ray tracing - slika

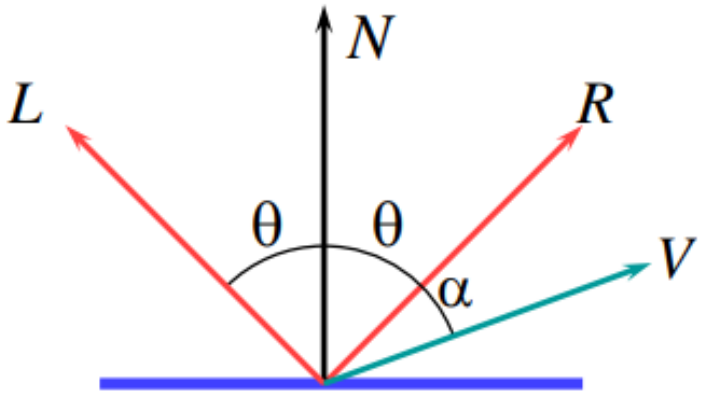
Diffuse Reflection



$$I = I_l k_d \cos\theta$$
$$= I_l k_d (N \cdot L)$$

L - normalizovan vektor ka izvoru svetlosti
N - normala na površinu
I_l - intenzitet svetlosti izvora
k_d - koeficijent difuznog odbijanja svetlosti od površine

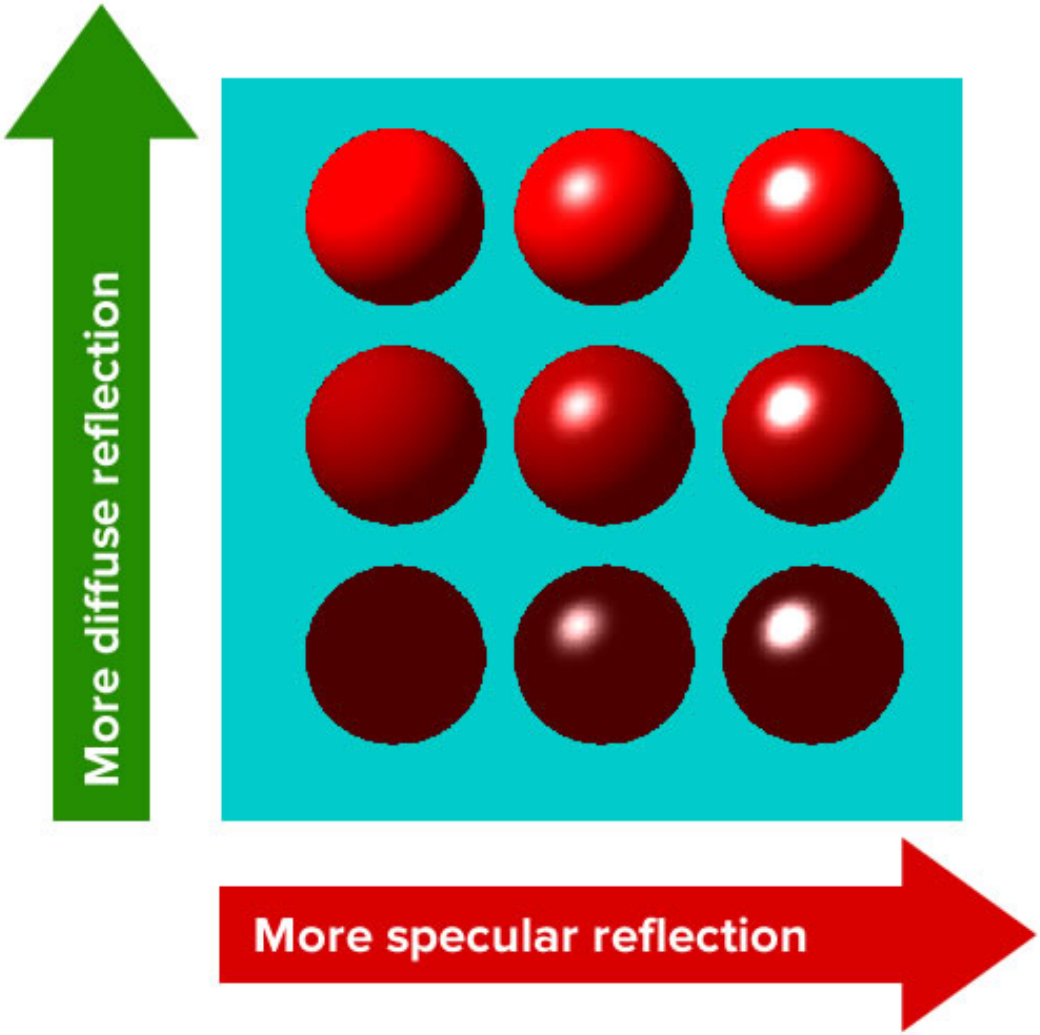
Specular reflection



$$I = I_l k_s \cos^n \alpha$$
$$= I_l k_s (R \cdot V)^n$$

R - smer savršene refleksije
k_s - koeficijent spekularnog odbijanja svetlosti od površine
n - Phong-ov koeficijent hrapavosti predmeta

Specular vs Diffuse reflection



Reflection formula

- Ambient reflection - homogeno distribuirano osvetljenje po celom prostoru
- Aproksimacija sekudarne refleksije od svih stvari

Formula ukupne osvetljenosti:

$$I = I_a k_a + \sum_i I_i k_d (L_i \cdot N) + \sum_i I_i k_s (R_i \cdot V)^n$$

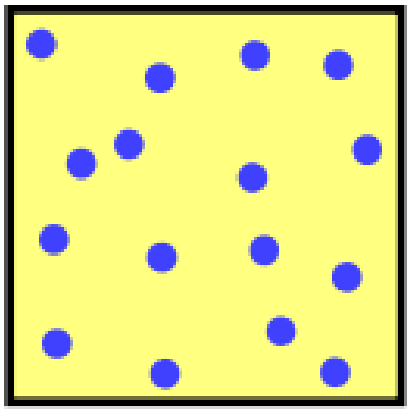
Antialiasing

- Pošto su pikseli diskretni, nama će ceo piksel biti jedne iste boje, i po sadašnjem algoritmu je to boja koje bi trebalo da bude centar piksela.
- Šta ako, kada bismo pustili više zraka kroz svaki piksel, oni ne bi bili iste boje?

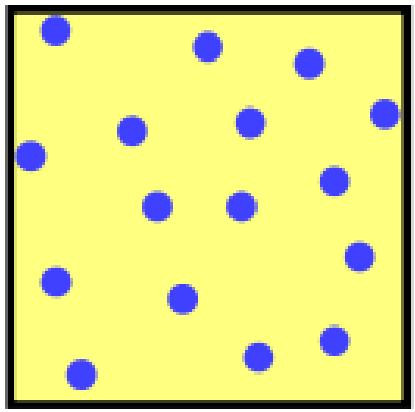
Antialiasing

- Pošto su pikseli diskretni, nama će ceo piksel biti jedne iste boje, i po sadašnjem algoritmu je to boja koje bi trebalo da bude centar piksela.
- Šta ako, kada bismo pustili više zraka kroz svaki piksel, oni ne bi bili iste boje?
- Najbolje je da ih na neki način uprosečimo.

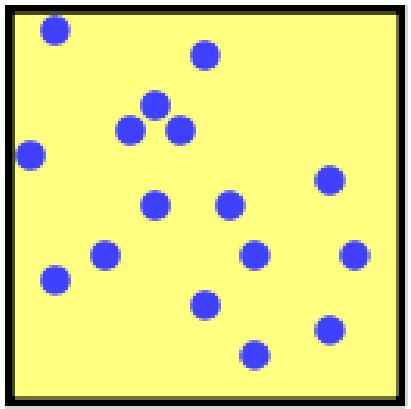
Nekoliko načina za uprosečavanje:



jittered



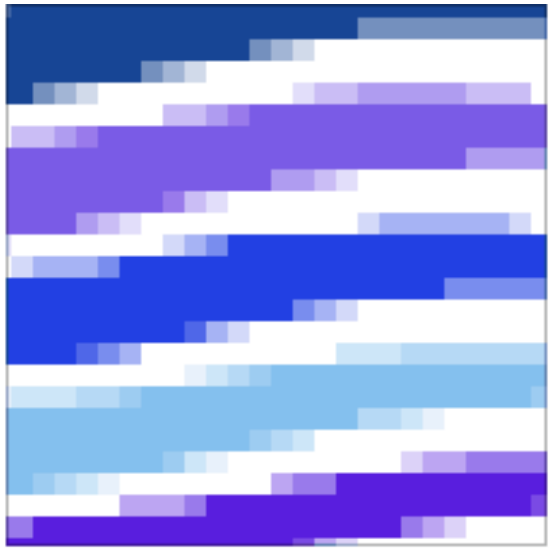
Poisson disc



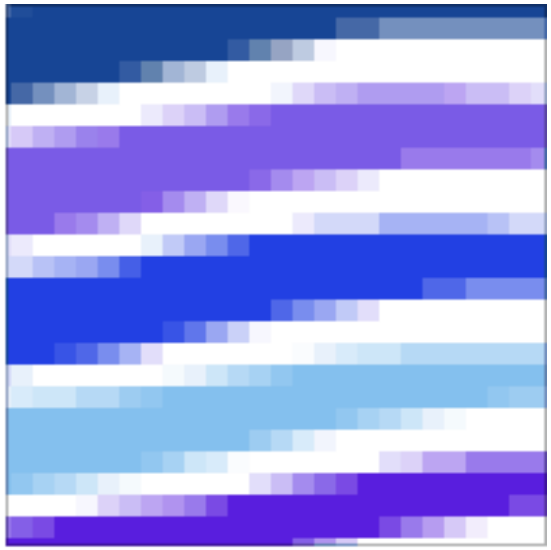
pure random

Antialiasing - primer

Anti-Aliasing = Off



Anti-Aliasing = 5 samples



Anti-Aliasing = 25 samples

Šta je Ray marching?

- Alternativa Ray tracingu
- Umesto da tražimo preseke našeg zraka sa ostalim objektima, pokušajmo da naš zrak propagiramo kroz prostor, i da vidimo u šta će da udari
- **Za koliko smemo nesmetano da se pomerimo, kada smo u nekoj tački prostora?**

Šta je Ray marching?

- Alternativa Ray tracingu
- Umesto da tražimo preseke našeg zraka sa ostalim objektima, pokušajmo da naš zrak propagiramo kroz prostor, i da vidimo u šta će da udari
- **Za koliko smemo nesmetano da se pomerimo, kada smo u nekoj tački prostora?**
- Uvedimo takvu funkciju..

Signed Distance Fields

- Idealno - najmanje rastojanje do bilo kog objekta na sceni.
- Teško izračunati idealno, ali bilo koja funkcija koja ni u jednoj tački ne premašuje rastojanje do najbližeg objekta radi.
- Što je naša funkcija bliža pravoj, to algoritam brže radi.
- Primer: Signed Distance Field za sferu?

Signed Distance Fields

- Idealno - najmanje rastojanje do bilo kog objekta na sceni.
- Teško izračunati idealno, ali bilo koja funkcija koja ni u jednoj tački ne premašuje rastojanje do najbližeg objekta radi.
- Što je naša funkcija bliža pravoj, to algoritam brže radi.
- Primer: Signed Distance Field za sferu?

```
float sphere(vec3 p, float r) {
  return length(p) - r;
}
```

Signed Distance Fields - još neki primeri

Kocka?

Signed Distance Fields - još neki primeri

Kocka?

```
float cube(vec3 p, vec3 dim) {
  vec3 d = abs(p) - dim;
  return min(max(d.x,
      max(d.y, d.z)), 0.0)
      + length(max(d, 0.0));
}
```

Valjak?

Signed Distance Fields - još neki primeri

Kocka?

```
float cube(vec3 p, vec3 dim) {
  vec3 d = abs(p) - dim;
  return min(max(d.x,
      max(d.y, d.z)), 0.0)
      + length(max(d, 0.0));
}
```

Valjak?

```
float cylinder(vec3 p, vec3 dim)
{
  return length(p.xz - dim.xy)
      - dim.z;
}
```

Torus?

Signed Distance Fields - još neki primeri

Kocka?

```
float cube(vec3 p, vec3 dim) {
  vec3 d = abs(p) - dim;
  return min(max(d.x,
      max(d.y, d.z)), 0.0)
      + length(max(d, 0.0));
}
```

Valjak?

```
float cylinder(vec3 p, vec3 dim)
{
  return length(p.xz - dim.xy)
      - dim.z;
}
```

Torus?

```
float torus(vec3 p, vec2 t) {
  vec2 q = vec2(
      length(p.xz) - t.x, p.y);
  return length(q) - t.y;
```

Kombinovanje SDF-ova

- Pokazali smo kako se za neke proste geometrijske oblike SDF dobija u nekoliko linija koda – šta da radimo kada želimo komplikovanije oblike?

Kombinovanje SDF-ova

- Pokazali smo kako se za neke proste geometrijske oblike SDF dobija u nekoliko linija koda – šta da radimo kada želimo komplikovanije oblike?
- Kombinujemo ih!
- Ukoliko imamo dva SDF-a, kako dobijamo njihovu uniju?

Kombinovanje SDF-ova

- Pokazali smo kako se za neke proste geometrijske oblike SDF dobija u nekoliko linija koda – šta da radimo kada želimo komplikovanije oblike?
- Kombinujemo ih!
- Ukoliko imamo dva SDF-a, kako dobijamo njihovu uniju?
- Presek?

Kombinovanje SDF-ova

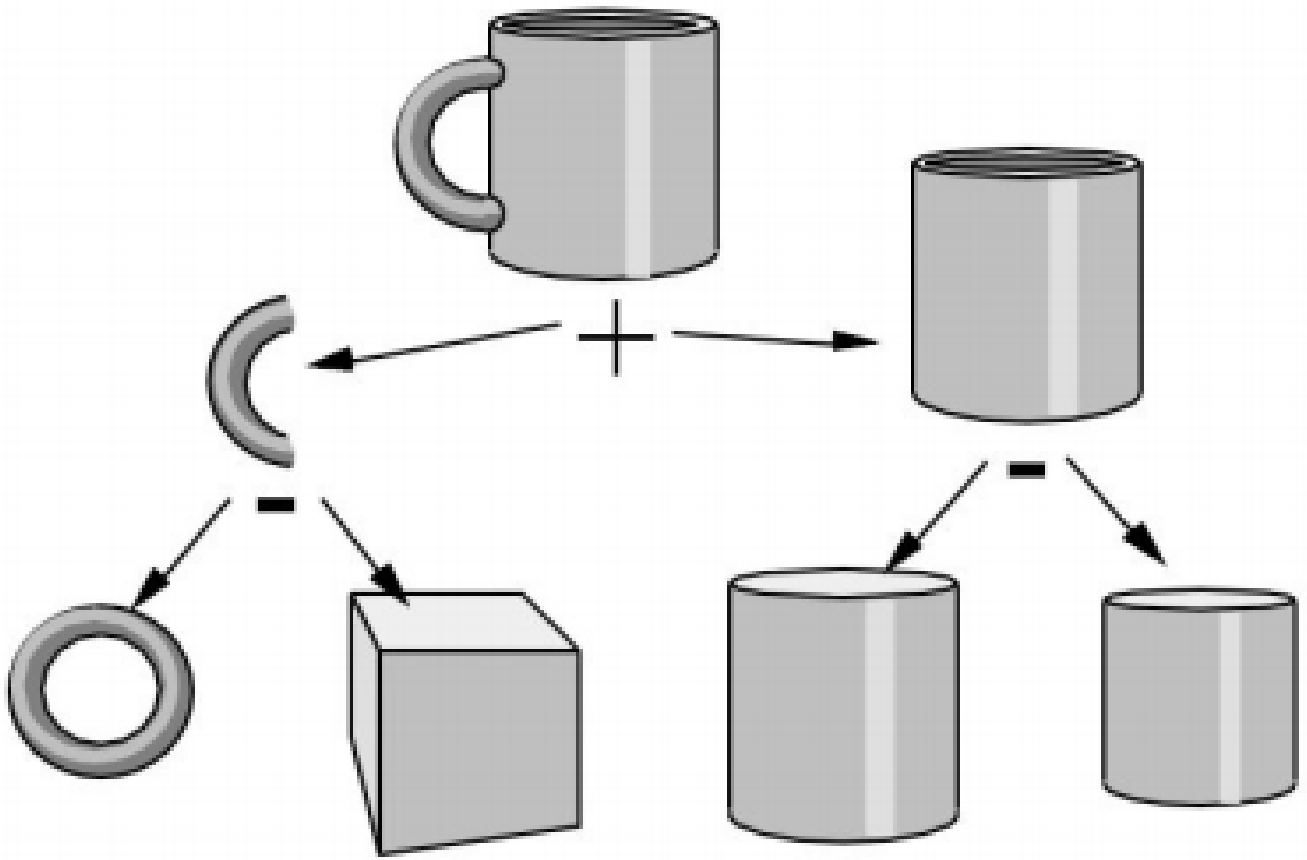
- Pokazali smo kako se za neke proste geometrijske oblike SDF dobija u nekoliko linija koda – šta da radimo kada želimo komplikovanije oblike?
- Kombinujemo ih!
- Ukoliko imamo dva SDF-a, kako dobijamo njihovu uniju?
- Presek?
- Razliku?

Kombinovanje SDF-ova

- Pokazali smo kako se za neke proste geometrijske oblike SDF dobija u nekoliko linija koda – šta da radimo kada želimo komplikovanije oblike?
- Kombinujemo ih!
- Ukoliko imamo dva SDF-a, kako dobijamo njihovu uniju?
- Presek?
- Razliku?

```
#define unite(sdf1, sdf2) min(sdf1, sdf2)
#define intersect(sdf1, sdf2) max(sdf1, sdf2)
#define difference(sdf1, sdf2) max(sdf1, -sdf2)
```

Kombinovanje SDF-ova - primer



Ray marching - algoritam

```
vec3 raymarch(vec3 pos, vec3 raydir) {
  int step = 0;
  float d = getSdf(pos);

  while (abs(d) > 0.001 && step < 50) {
    pos = pos + raydir * d;
    d = getSdf(pos);  // Return sphere(pos) or any other
    step++;
  }

  return
      (step < 50) ? illuminate(pos, rayorig) : background;
}
```

Normala na SDF

- Kako najlakše možemo izračunati normalu na SDF, ako imamo samo funkciju?

Normala na SDF

- Kako najlakše možemo izračunati normalu na SDF, ako imamo samo funkciju?
- Normala je smer u kom SDF najbrže raste:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

Kako izračunati?

```
float d = getSdf(pt);
vec3 normal = normalize(vec3(
    getSdf(vec3(pt.x + 0.0001, pt.y, pt.z)) - d,
    getSdf(vec3(pt.x, pt.y + 0.0001, pt.z)) - d,
    getSdf(vec3(pt.x, pt.y, pt.z + 0.0001)) - d));
```

Transformacije SDF-ova

- Ako želimo, da rotiramo, transliramo, ili skaliramo SDF model, treba da primenimo inverznu transformaciju na ulaznu tačku u nasoj funkciji rastojanja.
- Primer: Ako želimo da renderujemo sferu na poziciji $(0, 3, 0)$:

```
float sphere(vec3 pt, float radius) {
  return length(pt) - radius;
}

float f(vec3 pt) {
  return sphere(pt - vec3(0, 3, 0));
}
```

SDF - cont'd

Stvari o kojim nemam vremena da pričam a jako lepo saradjuju sa SDF-ovima:

- Glatki prelazi izmedju objekata
- Ponavljajuća geometrija
- Stvari koje se uvrću

Šta smo naučili?

- Nekoliko algoritama za renderovanje slike - svaki ima svoje prednosti i mane
- Ovi algoritmi se danas sve više koriste - mogučnost paralelizacije, GPU Ray tracing, mnogo veći čomputing power"
- Uspeli smo da prodjemo srž svih algoritama koji se koriste već 50 godina u ove svrhe.

Recommended reading

- Shirley, P. Marschner, S. (2009). Fundamentals of Computer Graphics. CRC Press (3rd ed.)
- Watt, A. (1999). 3D Computer Graphics. Addison-Wesley (3rd ed).
- Hughes, van Dam, McGuire, Skalar, Foley, Feiner Akeley (2013). Computer Graphics: Principles Practice. Addison-Wesley (3rd edition)
- Rogers, D.F. Adams, J.A. (1990). Mathematical elements for computer graphics. McGraw-Hill (2nd ed.).

Hvala na pažnji!
Pitanja?